Le filtrage passé au crible

Stage de M1

Paul Patault

Université Paris-Saclay

22 juin 2022

Le filtrage, c'est quoi ?

- construction essentielle des langages fonctionnels
- raisonnement structurel par cas
- dont on cherche à vérifier l'exhaustivité et l'absence de redondance

```
let compare p = match p with
  | [], [] -> ...
  | [], _  -> ...
  | _,  [] -> ...
  | E :: _, E :: _ -> ...
  | N (E, _) :: _, N (E, _) :: _ -> ...
  | N (_, _) :: _, _ -> ...
  | _, N (_, _) :: _ -> ...
```

Gospel : Generic OCaml SPEcification Language

- langage de spécification formelle pour OCaml
- logique du premier ordre (pas nécessairement exécutable)
- sémantique basée sur la logique de séparation

```
val find_opt : ('a -> bool) -> 'a list -> 'a option
(** [find_opt f l]
    renvoie [Some x] si [x ∈ l] et [f x]
            [None] sinon *)
(*@ r = find_opt f l
      ensures match r with
        | None -> List.for_all (fun x -> not (f x)) l
        | Some x -> f x /\ List.mem x l *)
```

https://www.github.com/ocaml-gospel/gospel

Motifs

Un motif est défini par la syntaxe abstraite suivante :

$$
\begin{array}{rcll}
m & ::= & & \\
& | & \_ & \textit{Attrape-tout} \\
& | & x & \textit{Variable} \\
& | & C(m, \ldots, m) & \textit{Constructeur} \\
& | & m \,|\, m & \textit{Motif-ou}
\end{array}
$$

Relation de filtrage

Les règles d'inférences suivantes définissent une relation de filtrage sur les motifs, notée m $\preceq$ m' :

$$\frac{}{\_ \preceq m}\ (\text{Def}) \qquad \frac{}{x \preceq m}\ (\text{Var})$$

$$\frac{m \preceq m'_1 \qquad m \preceq m'_2}{m \preceq m'_1 \mid m'_2}\ (\text{Or}_0) \qquad \frac{m_1 \preceq m'}{m_1 \mid m_2 \preceq m'}\ (\text{Or}_1) \qquad \frac{m_2 \preceq m'}{m_1 \mid m_2 \preceq m'}\ (\text{Or}_2)$$

$$\frac{\forall i.\ i \in [1..k] \Rightarrow m_i \preceq m'_i}{C(m_1, \ldots, m_k) \preceq C(m'_1, \ldots, m'_k)}\ (\text{Constr})$$

Matrice de filtrage

$M_{m \times n}$ filtre n valeurs avec m motifs

```
let f x =
  match x with
  | e,         []    -> 1
  | Some _,    [42]  -> 2
  | _,         h::t  -> 3
```

$$\begin{pmatrix} \texttt{e} & \texttt{[]} \\ \texttt{Some}\ \_ & \texttt{[42]} \\ \texttt{\_} & \texttt{h::t} \end{pmatrix}$$

- la matrice M filtre un vecteur de motifs $\vec{q}$ (noté $M \preceq q$) si et seulement si $\exists i.\ \forall j.\ M_{i,j} \preceq q_j$
- la matrice M est exhaustive si et seulement si $\forall\ \vec{q}.\ M \preceq \vec{q}$
- un vecteur $\vec{q}$ est utile à M si et seulement si $M \not\preceq \vec{q}$

Algorithme `usefulness` (Maranget 2003)

```
usefulness : mat -> vec -> bool
```

- l'appel `usefulness` M $\vec{q}$ renvoie « $\vec{q}$ est utile à M »

↳ M est exhaustive si et seulement si `usefulness` M $\vec{q}$ = `false` avec $\vec{q}$ = `(_, ..., _)`

↳ la ligne M_i est redondante si et seulement si `usefulness` M' $\vec{q}$ = `false` avec $\vec{q} = M_i$ et $M' = \begin{pmatrix} M_1 \\ \vdots \\ M_{i-1} \end{pmatrix}$

Indécidable ?

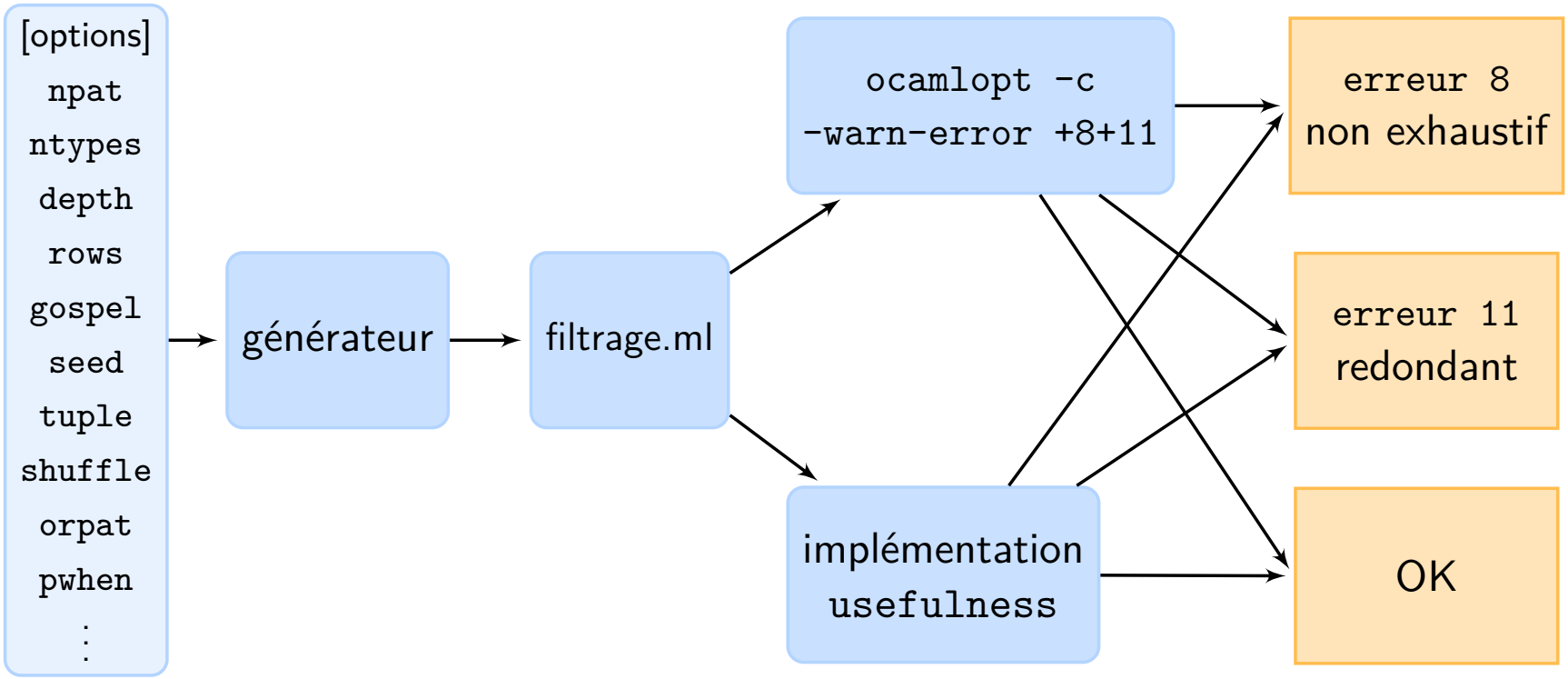
- les `when` permettent une écriture élégante :

```
let f t = match t with
  | N _ when ... -> ...
  | N (N _, _) when ... -> ...
  | E | N (E, _) | N (_, E) -> ...
```

- mais le problème de l'exhaustivité devient indécidable (thm. Rice)
- on peut cependant parfois montrer que le filtrage est exhaustif avec une réduction vers le problème initial

$$\begin{pmatrix} \texttt{N \_} & \texttt{true} & \texttt{\_} \\ \texttt{N (N \_, \_)} & \texttt{\_} & \texttt{true} \\ \texttt{E | N (E, \_) | N (\_, E)} & \texttt{\_} & \texttt{\_} \end{pmatrix}$$

Générateur aléatoire de problèmes de filtrage



+10 000 problèmes de filtrage vérifiés (∼500 000 lignes)

Générateur aléatoire de problèmes de filtrage

```
./genpat -pwhen 0 -npat 1 -tuple 1 -typ-args 2 -dmax 3 -lmax 5 -seed 144
```

```
type t = A of t * t | D of t * t | B of int * int
let _ = fun x -> match x with
  | B(_, 8) -> ()
  | D(D(B(13, _), B(21, _)), D(A(_, _), A(_, _))) -> ()
  | D(A(B(_, 34), _), D(B(55, 89), A(_, _))) -> ()
  | A(D(A(_, _), A(_, _)), _) -> ()
  | A(A(_, _), _) -> ()
```

En résumé

autour de l'algorithme `usefulness` :

- implémentation dans Gospel
- vérification du code par génération aléatoire de tests
- preuve de correction, terminaison et complexité
- support des clauses `when`

suite du stage :

- « gospelisation » d'OCamlGraph
- contributions à l'implémentation de Gospel (12 PR acceptées)